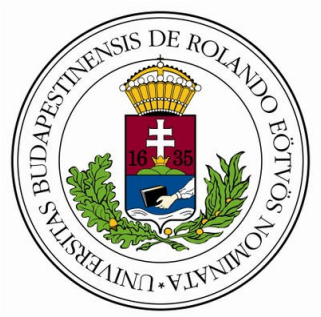


Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs Rendszerek Tanszék

Hierarchia mentén aggregált adatokat kezelő prototípus alkalmazás

Dr.Vincellér Zoltán
Mesteroktató

Kaszner Máté
Programtervező Informatikus BSc

Budapest, 2020

Tartalomjegyzék

1. Bevezetés

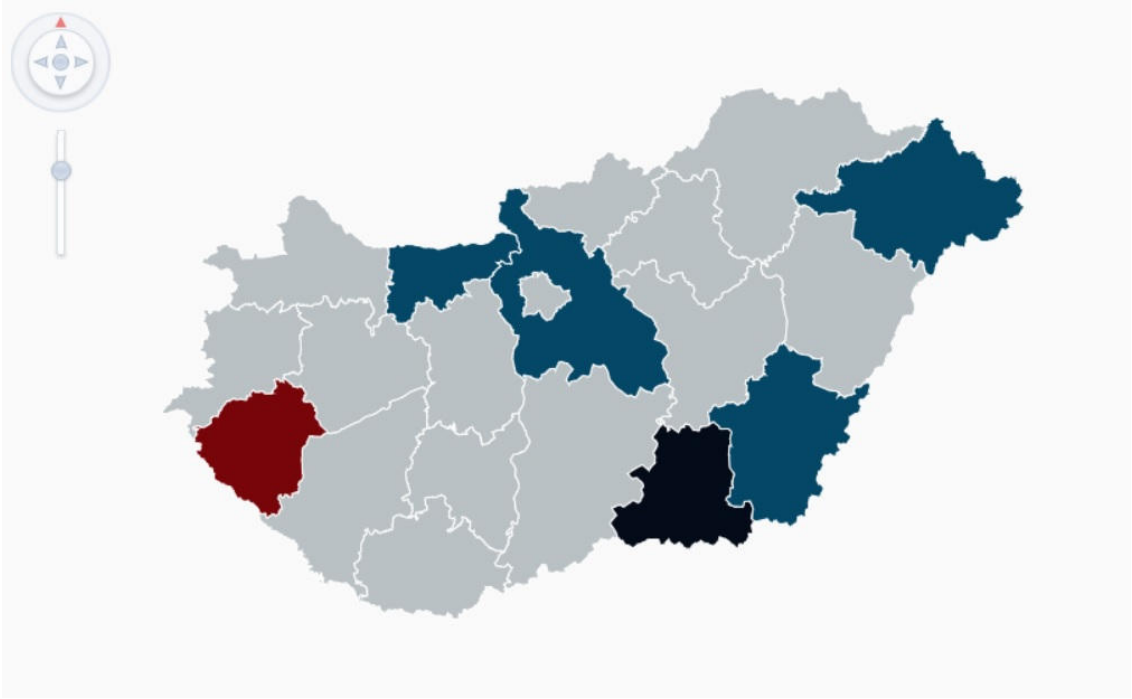
1.1. Feladat

Backend oldali nagy tömegű analitikus adatok összetett hierarchiák mentén történő aggregálásának, továbbá a összegzett tematikus térképi-, valamint részletes analitikus-adatok frontend oldali vizualizációjának elkészítése (például táblázat, illetve diagramok felhasználásával). Az elkészült alkalmazás egy prototípusa lesz az ilyen nagymennyiségű hierarchikus adatok backend, illetve frontend oldali feldolgozásának. Az applikáció fő célja, hogy minnél látványosabban ábrázolja az egyes hierarchiák mentén történő lefúrást.
A feldathoz adott volt, hogy a megoldásához az SAP Cloud Platform webes környezetét használjam, új módszereket és megvalósítási lehetőségeket találva, ezen a nemsokak által használt és viszonylag új környezetben, véleményem szerint. Kezdetben csak a fő csoportok jelennek meg, majd ezek kiválasztásával egyre mélyebb szint jelenik meg párhuzamosan a táblázatos, diagramos, illetve a térképes nézeten. Amikor a legmélyebb szintre értünk, akkor kiválasztva az adott elemet, megjelenik a térképen a jelzője, amire a térképes nézet rá is közelít. Ahhoz, hogy letudjunk fúrni az egyes szinteken, nemcsak a táblázat használható erre a célra, hanem ugyanúgy, ha a digramban/térképen kattintunk a lefúrni kívánt csoportra, akkor azt párhuzamosan követi a táblázat és a térkép/diagram. A térképen Magyarország térképe látható, megyék alapján felosztva.

1.2. Motiváció

A szakdolgozati téma kiválasztásakor nagy szerepet játszott az, hogy a szakmai gyakorlatom ideje alatt hasonló feladaton dolgoztam. A szakmai gyakorlat folyamán találkoztam először ezzel az úgymond új SAP Cloud Platform webes fejlesztői környezetével, ahol először meg kellett ismerkedni ezzel a felülettel. Ezek után a feladat az volt, hogy kutató munka jeleggel megoldást találjunk arra a problémára, hogy hogyan lehet megjeleníteni egy adathalmazt, amely egy adott hierarchia szerint aggregálható, majd ezt milyen eszközökkel lehet a leglátványosabb módon megjeleníteni. Ez a fajta munka, azért tetszett nekem, mert olyan megoldásokra kellett rájönni sokszor magamtól, amiket mások még nem használtak, vagy azt még nem osztották meg az internet közösségével. A gyakorlat alatt AngularJS-ben és SAPUI5-ban is implementálni kellett a feladatot, hogy következtetést tudjunk levonni, a kettő összehasonlítása után.
Az én választásom azért esett most az AngularJS-re, mert úgy láttam, több olyan funkció van még, amit nem használtam fel, amivel még bővíteni lehet ezt az ötletet, illetve sokkal több AngularJS alapú külső könyvtári elemet találtam, amit fel tudtam

használni a feladat megoldása során. A SAPUI5 nekem túlságosan kötöttnek tűnt, azaz például, ha egy táblázatba akartam betölteni az általam használt adatokat, akkor volt definiálva pár táblázat más-más paraméterekkel, de ha az én adatom egyikkel sem egyezett, akkor az adathalmazt kellett volna módosítani hozzá. A térkép implementálásával is megpróbálkoztam SAPUI5 és a Google Maps Javascript API felhasználásával. SAPUI5-ban is van egy előredefiniált analitikus térkép, amivel szintén sikerült megyénként felosztva ábrázolni Magyarország térképét, amit az 1.ábra mutat, de végül azért a Google Maps Javascript API mellett esett a választásom, mert funkciói terén is előnyösebbnek tartottam, illetve látvány szinten is jobban elnyerte a tetszésemet.



1. ábra. Sapui5 analitikus térkép, kijelölve rajta pár megyével

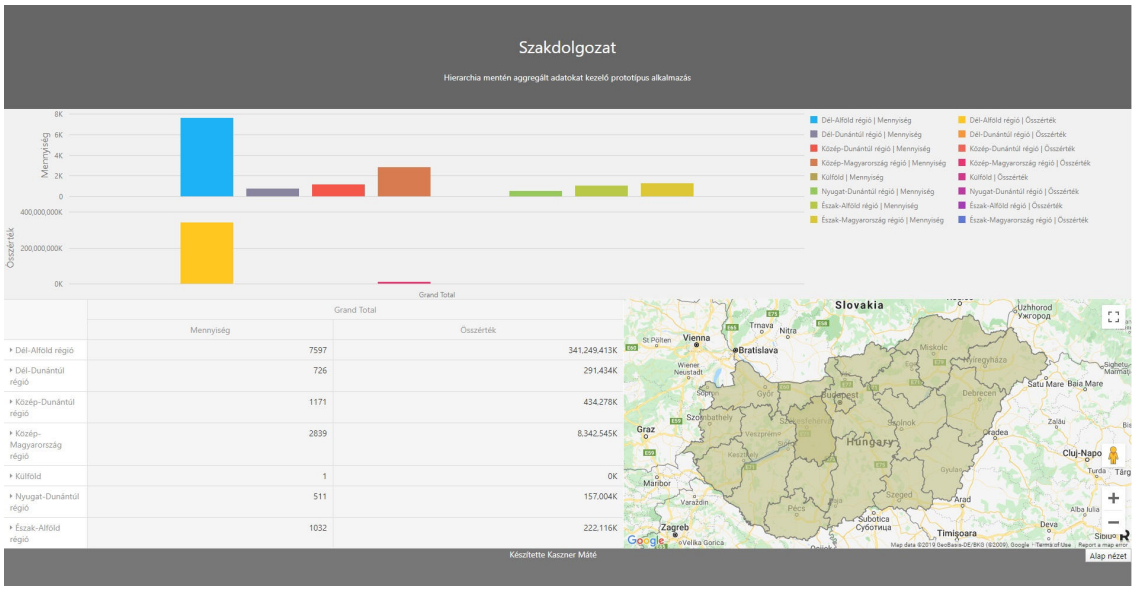
2. Felhasználói dokumentáció

2.1. A referencia feladat

Adottak Magyarország ingatlanjainak összesített listája településenként. Adott még a régió, a megye, amibe a település tartozik, az ingatlanok mennyisége, összesített értéke és fajtája, valamint még pár elhanyagolható adat. A feladat az volt, hogy ezeket az adatokat felhasználva, egy olyan hierarchikus adathalmazt hozzanak létre, amit ilyen módon meg is lehet jeleníteni. A vizualizáció táblázat, diagram és térkép segítségével történt, amelyek párhuzamosan, egymástól függően működnek, azaz egy elem kiválasztása bármely vizualizációs modulon, az a többire is hatással van. Ezek a modulok mind egy lapon helyezkednek el, ezzel fokozva a vizuális hatást.

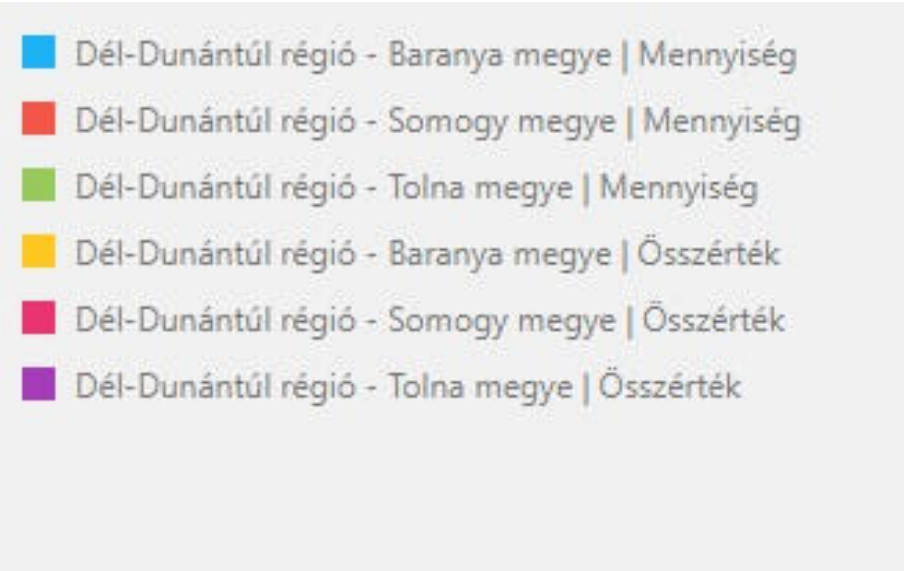
2.1.1. Megoldás

Az adatok megjelenítéséhez mind a 3 módszert, azaz a táblázatos, diagramos és térképes módszert is alkalmaztam. A táblázatban az egyes csoportok lenyitható fülként jelennek meg, tehát a táblázatnak az első oszlopában Magyarország régiói jelennek meg, a másik két oszlopában pedig az ingatlanok mennyisége és azok érteke forintban(ezres nagyságrendben) régiónként összesítve. Ekkor a diagramon szintén a régiók szerinti összesített adatok láthatók, a térképen pedig Magyarország térképe van kiemelve, ahol Magyarország megyéi vannak elhatárolva egymástól. Ezt a kezdeti állapotot a 2. ábra szemlélteti.



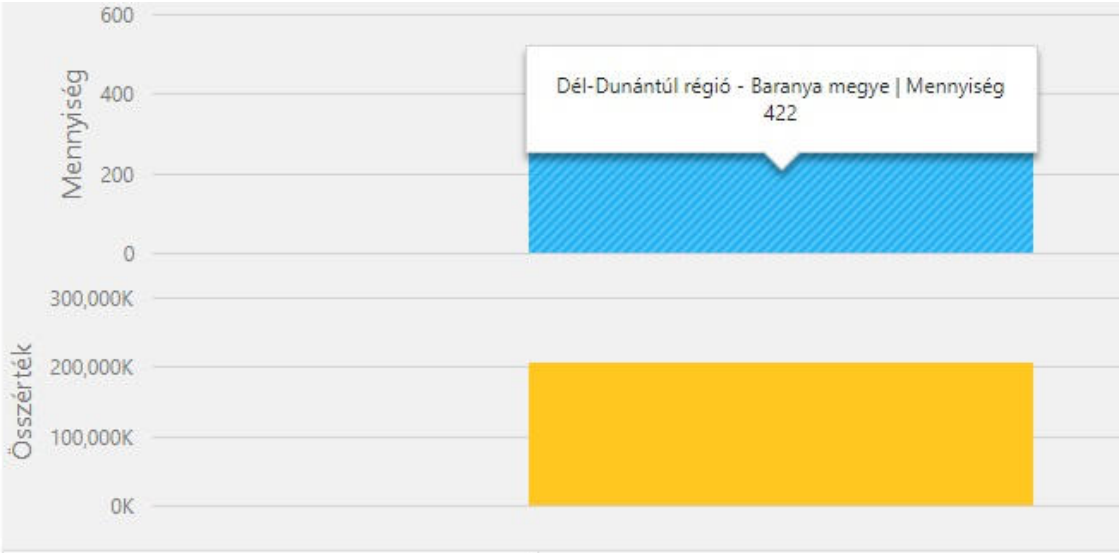
2. ábra. Régiónként összesített adatok.

A diagram tartalmaz a megjelenített adatok mellett egy jelmagyarázatot is, ami azt mutatja be, melyik szín milyen értéket jelöl. A leírás az elem nevét és az összesített érték típusát tartalmazza. Ezt a 3.ábra mutatja be.



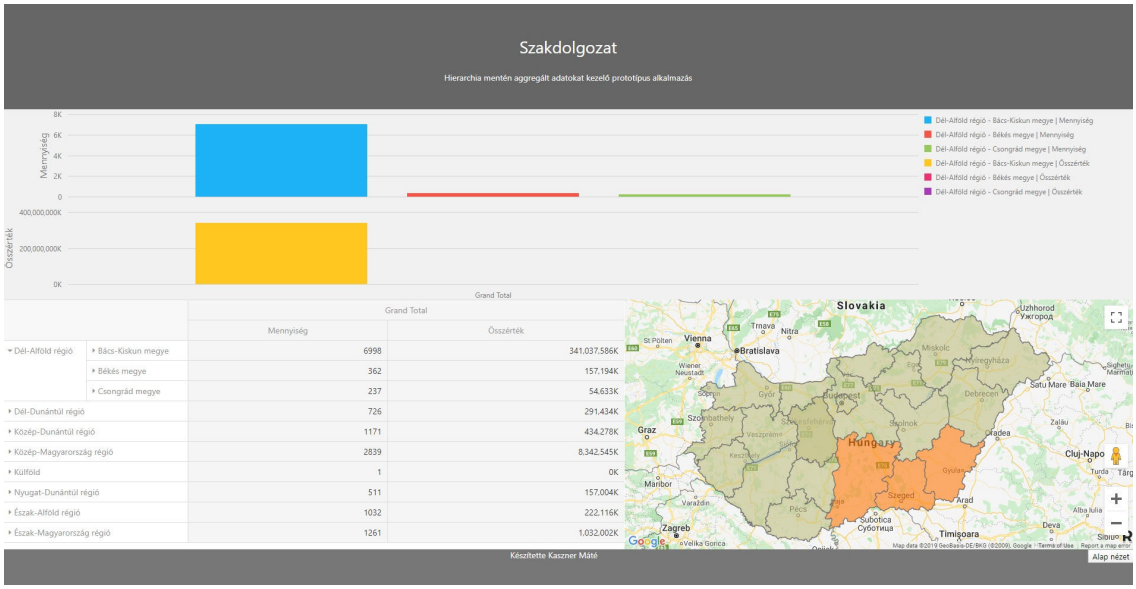
3. ábra. A diagram jelmagyarázata.

A diagram további funkciói közé tartozik az is, amikor a felhasználó az oszzlopdiagram egyik elemére visszi a kurzort, akkor egy kis információs ablak jelenik meg az oszlop felett. Ez az ablak jelzi az adott elem nevét, az összesített érték típusát(mennyiség vagy összérték) és a konkrét értéket. Az elem neve állhat több részből, amiket kötőjel választ el, ugyanis attól függ, hányadik szintig lett lefúrva az értékekben. A 4.ábrán megye szinten van ez a kiegészítő bemutatva.



4. ábra. A diagram elemének információs mezője.

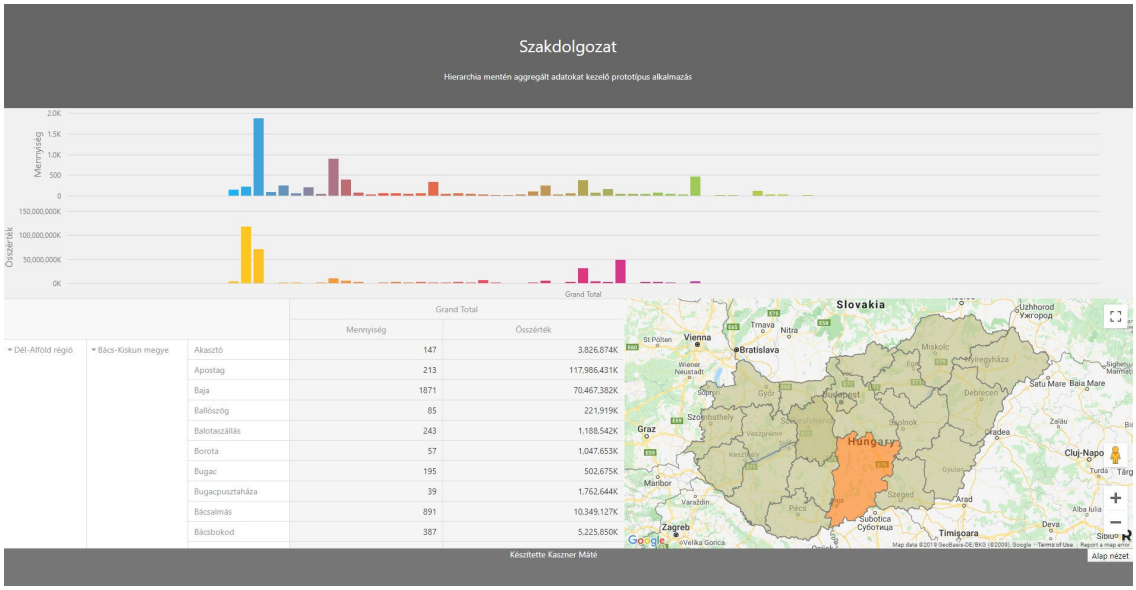
Amennyiben a táblázatban a felhasználó kiválasztja az egyik régiót, azaz az egyik lenyitható fülére kattint, akkor a táblázatban a kiválasztott régió megyéi is megjelennek az első oszlop másik felében, amik ugyanúgy lenyitható gombokként valósulnak meg, mellettük pedig az adatok már megyénként vannak összesítve. Ezzel egy időben a diagramon is az adott régión belüli megyék összesített adatai jelennek meg, a térképen pedig narancssárga színt kapnak azok a megyék, amelyek a kiválasztott régióba tartoznak. Abban az esetben, ha a diagramban választja ki az egyik régiót a megjelenített adatok közül, akkor az szintén hatással van a táblázatos és térképes nézetre egyaránt, tehát a táblázatban a kiválasztott régió lenyílik, valamint a térképen is az adott régióba tartozó megyék válnak kijelöltté. Mindezt a 5.ábra ábrázolja. Ellentétes irányban, a táblázatban egy régió bezárásakor a diagramon újra a régiók szerinti összegzés lesz látható, a térképen pedig megszűnik a régió kijelölése.



5. ábra. Régió lenyitva, megyénként összesített adatok.

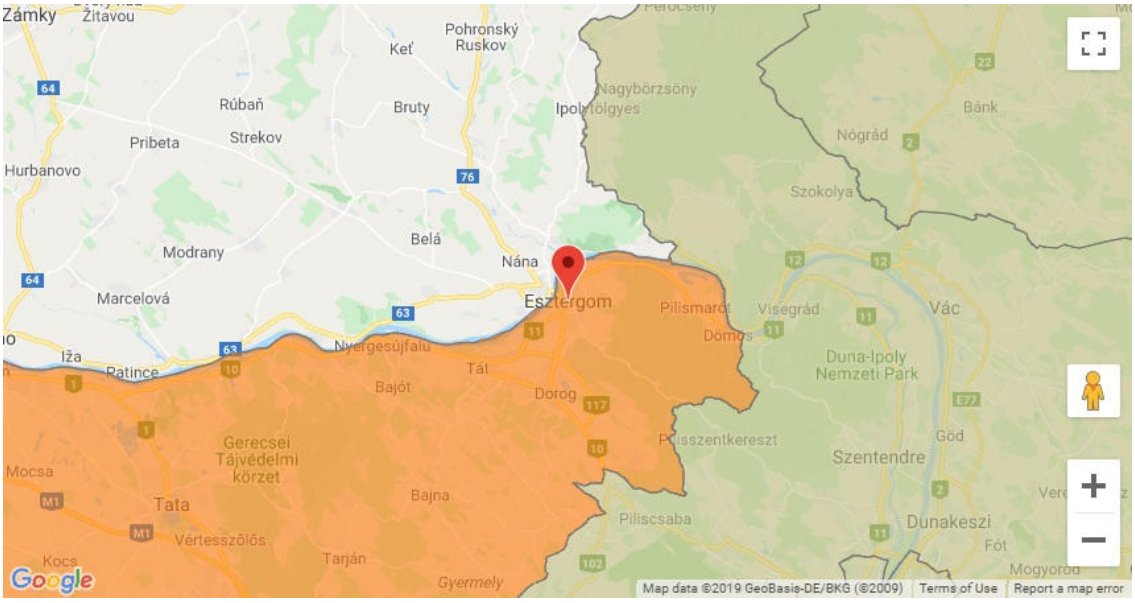
A kapott értékek még tovább részletezhetők, azaz egy adott régión belüli megyék egyikére kattintva, megjelenik a kiválasztott megye települései szerinti értékek összegzése. A táblázatban az első nagy oszlop harmadik részében láthatók ezek a települések, mellettük pedig az összesített mennyiség és az értékük forintban számolva. A táblázat mikor már kellően sok adatot tartalmaz ahhoz, hogy már nem lehetséges a táblázat eredeti méretében megjeleníteni, akkor ugyan a táblázat mérete megmarad, de ezúton görgethetővé válik. A diagramon szintén elérhető e funkció, tehát ha a diagramon egy régió megyénkénti összegzése látható, akkor azok bármelyikére kattintva, ugyanúgy megjelenik a kiválasztott megye településenkénti szétírása. Viszont ekkor már a diagram melletti színenkénti jelmagyarázat eltűnik, mivel a települések

száma megyénként kellően sok ahhoz, hogy mindhez jelmagyarázat tartozzon. A térképen ekkor már csak az utoljára kiválasztott megye van megjelölve. Ezt a 6.ábra szemlélteti. Fordított esetben, a táblázatban egy megyét becsukva,a térképen a régió megyéi jelennek meg kijelölve, amibe a becsukott megye tartozik, a diagramon pedig a régió megyénkénti felosztása kap helyet.



6. ábra. Megye lenyitva, településenként összesített adatok.

A térképes nézet mivel kezdetben is megyénként felosztva ábrázolja Magyarországot, ezért a térképen a felhasználó akár egyből egy megye szintre tud lefúrni. Amennyiben a kiválasztott megye olyan régióba tartozik, amely korábban már le-volt nyitva, akkor egyből az adott megye településenkénti összegzése jelenik meg a táblázaton és a diagramon egyaránt. Azonban ha a régió még nem volt lenyitva, akkor a térképen a megyére kétszer kattintva egymás után, először lenyílik a régió, majd utána a megye. Mikor már három szintet lefúrt a felhasználó a vizualizált adatokban(Régió - Megye - Település), a táblázat és a diagram nézet tovább már nem bontható, de amennyiben a táblázatban vagy a diagramon egy településre kattint, akkor a térképen ráközelítve megjelenik az adott település egy jelző(marker) formájában. Ez látható a 7.ábrán a következő oldalon.



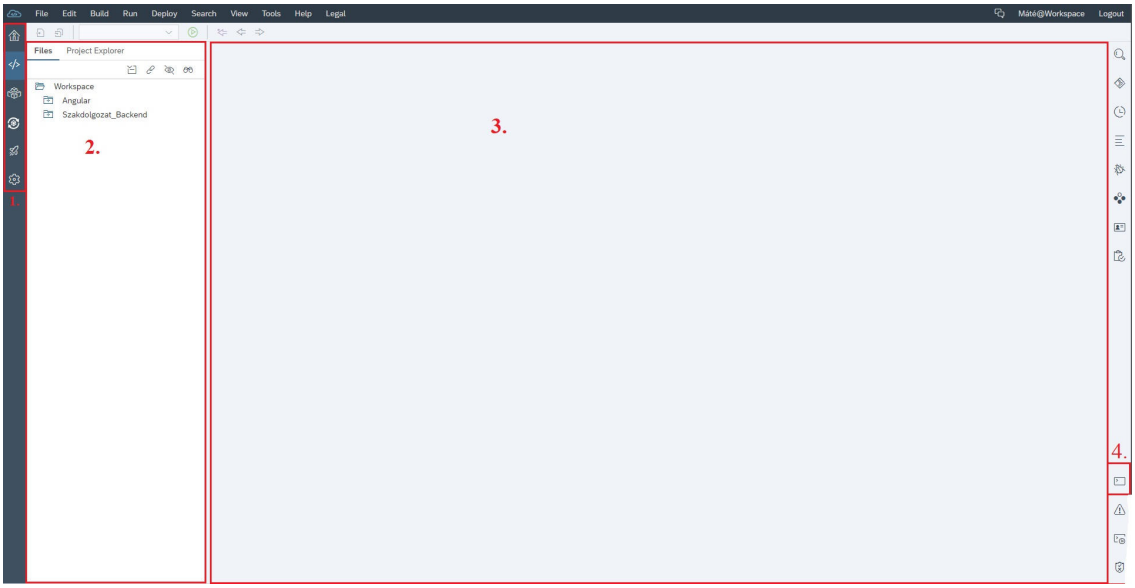
7. ábra. Térképen az Esztergom település megjelölve.

Az oldalon látható egy gomb is Alap nézet felirattal, ami arra szolgál, hogy amikor a táblázatban a felhasználó rákattint egy vagy akár több településre és a térképen az(azok) ráközelítve meg is jelenik, ekkor a térképet visszahelyezi alap nézetbe, letörölve róla a jelölőket és visszállítva a középpontba Magyarország térképét. A gomb lenyomása nincs hatással a táblázatra és a diagramra, azok a gomb lenyomás előtti állapotban maradnak. A térképen a kijelölés szintén megmarad, csak a piros jelölő tűnik el.
Az alkalmazás vizualizációs moduljait egy fejléc és egy lábléc hátárolja. Az alap nézet gomb is ezen a szürke láblécen folglal helyet, közvetlenül a térkép alatt funkcionalitásbeli kapcsolat miatt.

2.2. Futtatási környezet bemutatása

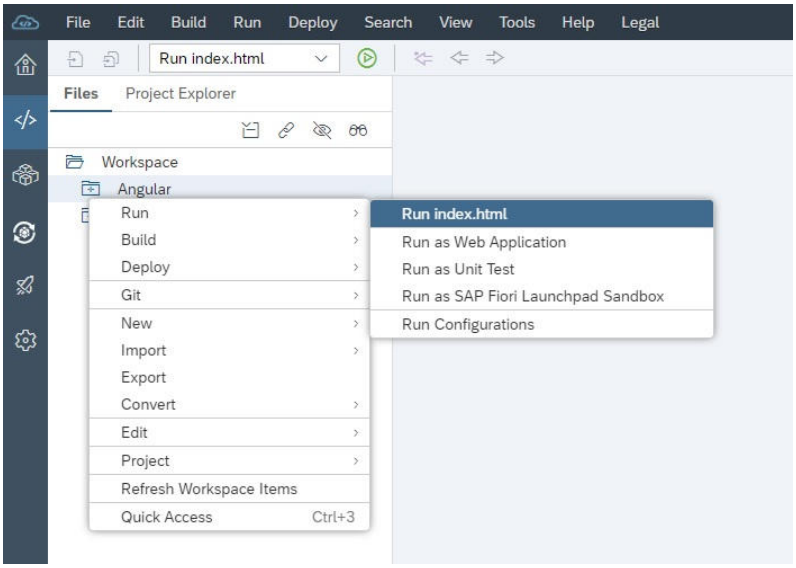
Az applikációt az SAP Cloud Platform webes fejlesztői felületén(WebIDE) tudja a felhasználó futtatni. A webes fejlesztői környzezet nagyrészét az úgynevezett Cloud Foundry biztosítja, ugyanis ez felel az egyes fájlok,komponensek vagy modulok fordításáért és futtatásáért. A fejleszetői környzet többi része a Neo Trial irányítása álá tartozik.
Belépve a fejleszetői környzetbe a következő egységek láthatók a felhasználó számára: (1)Az oldal bal szélén az alap és a felhasználó által hozzáadott kiegészítők találhatók(mint például Development, Database Explorer, Preferences, stb.). (2)A baloldali menüsáv mellett a felhasználó saját munkaterülete(Workspace) foglal helyet, amiben a saját projektjei helyezkednek el. (3)Az odlal nagyrészén maga a fej-

lesztői felület van, ahol megjelenik a szerkeszteni kívánt fájl. (4)A jobb alsó sarokban található a konzol ablak megnyitására szolgáló ikon. Ezek az elemek a 8.ábrán kijelölve láthatók.



8. ábra. SAP Cloud Platform Web IDE környezete.

Az applikáció futtatásához a munkaterületében található frontend oldali Angular nevű projektre kell kattintani jobb egér gombbal, majd a Run - Run Index.html opciót kell kiválasztani, amit a 9.ábra mutat. Egy sikeres bejelentkezést követően el is indult az applikáció.

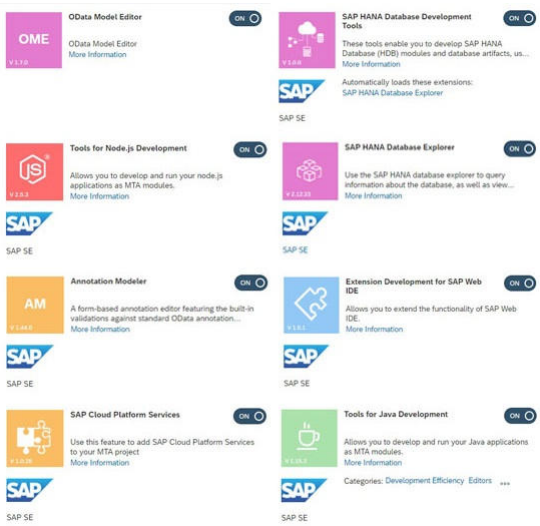


9. ábra. Az alkalmazás futtatása.

2.2.1. Telepítés

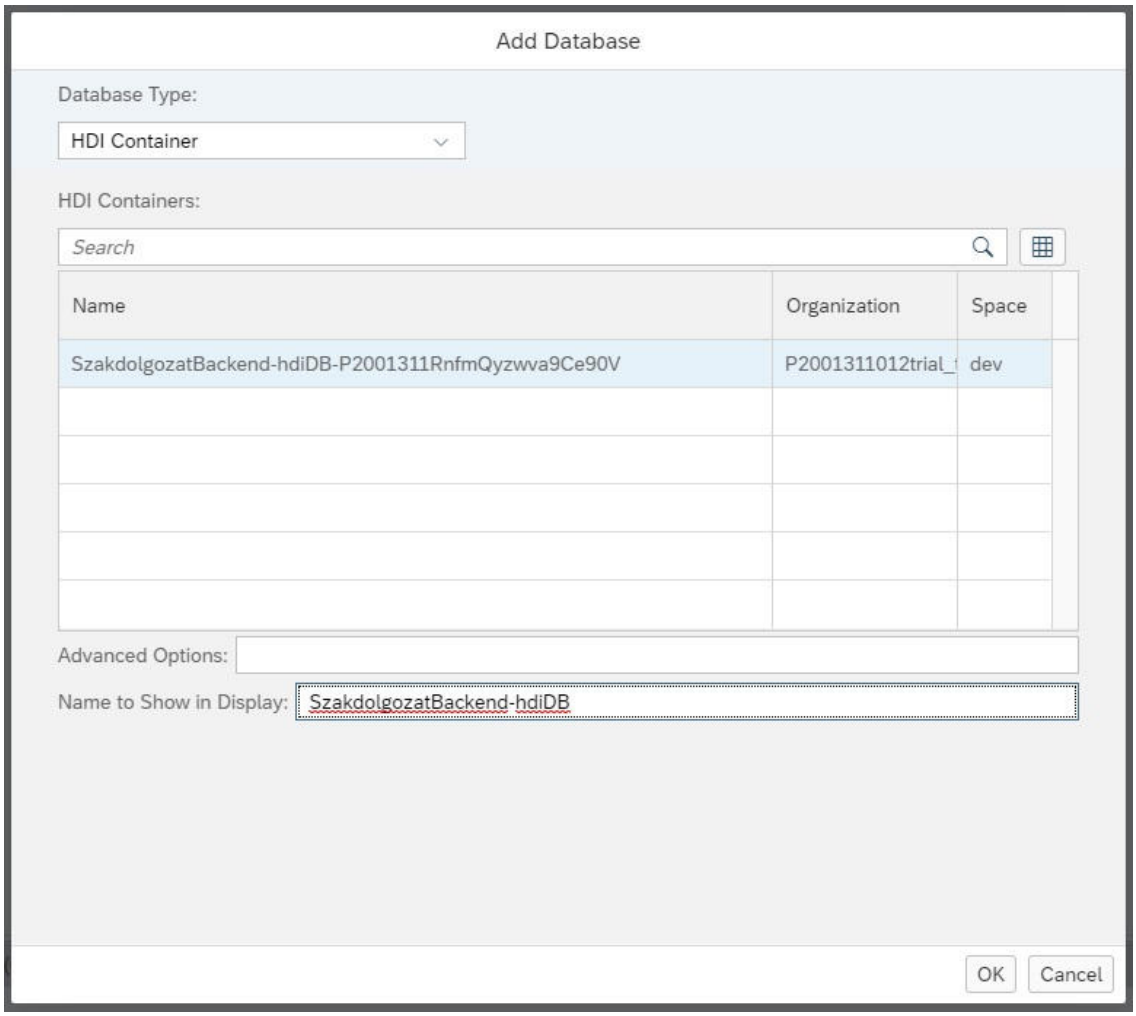
Az applikáció telepítéséhez szükség van egy SAP Cloud Platform felhasználói fiókra. A bejelentkezés után a főoldalon az Enviroments címsor alatt található egy "Access Cloud Foundry Trial" link, ahol igénybe kell venni a Cloud Foundry területet(Európai szervert kiválasztva). Ezután visszalépve a főoldalra ott található a "Launch SAP Web IDE" gomb, ezzel elindítva a webes fejlesztői környezetet.
A Web IDE bal oldali menüjéből a "Preferences" gombra kattintva megnyílik az a felület, ahol a Cloud Foundry beállításánál ki kell választani a nemrég létrehozott Cloud Foundry területet, majd ezután menteni a beállítást. Ezek után az "Extensions" (kiegészítők) lehetőséget kiválasztva be kell kapcsolni a következő kiegészítőket, amiket a 10.ábra is jelöl:

- OData Model Editor
- SAP HANA Database Development Tools
- Tools for Node.js Development
- SAP HANA Database Explorer
- Annotation Modeler
- Extension Development for SAP Web IDE
- SAP Cloud Platform Services
- Tools for Java Development



10. ábra. A szükséges kiegészítők.

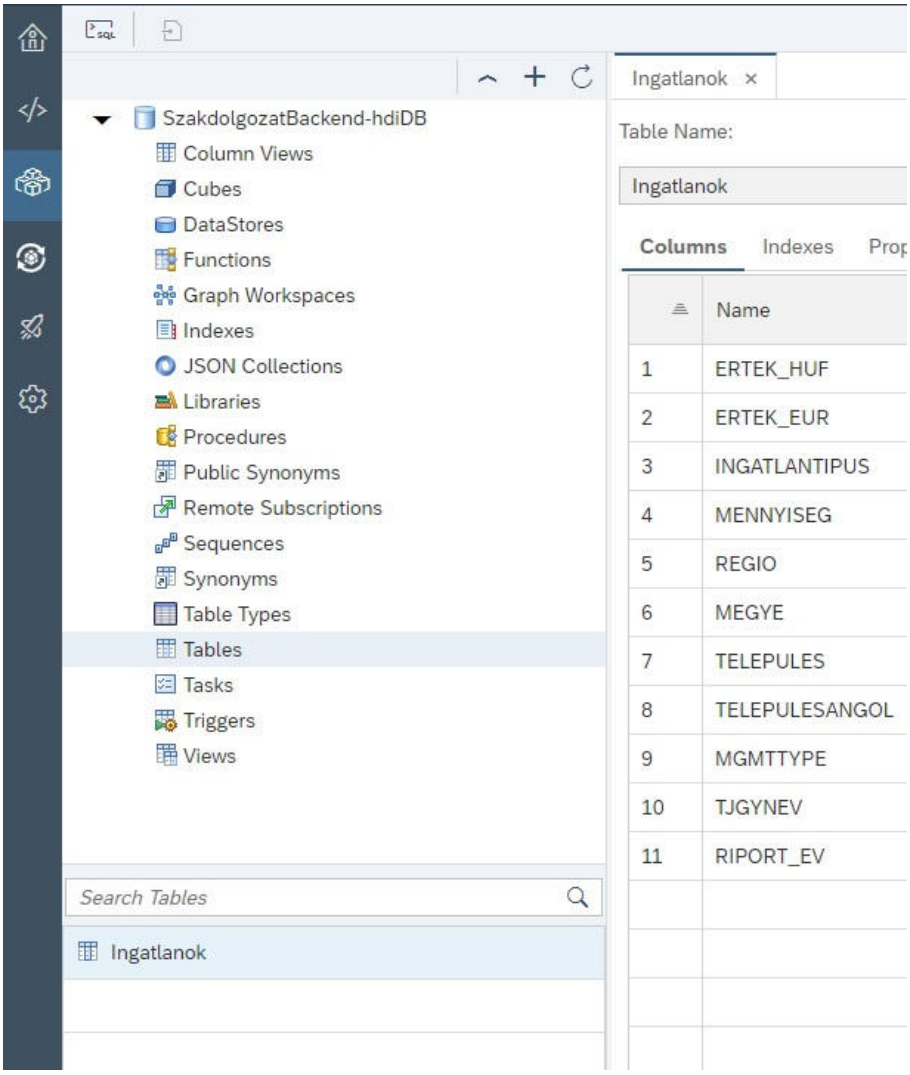
A baloldali menüben a Development gombra kell kattintani. A Workspace-re jobb egérgombbal kattintva kell beimportálni mindkét projektet(Szakdolgozat_backend, Angular). A backend projekten belüli DB mappára jobb egérgombbal kattintva válasszuk ki a "Build" parancsot. A fordítás vége a konzolban lesz feltüntetve.
A baloldali menü segítségével át kell lépni a Database Explorer részbe, ahol a "+" ikonra kattintva (Add a database to the Database Explorer) a felugró ablakban(11.ábra) a backend projekt nevéhez hasonló hdiDB-t kell kiválasztani.



11. ábra. A + gombra kattintva megjelenő ablak.

Az újonnan létrehozott HDI Container-en belül meg kell nyitni a "Tables" opciót, majd az Ingatlanok táblára jobb egérgombbal kattintva beimportálni az adatot. Ki kell választani a megfelelő .csv kiterjesztésű fájlt és betölteni. Azt, hogy a .csv fájl melyik oszlopából a tábla melyik oszlopába kerül a megfelelő adat, ezt eldönti magától a fejlesztői környezet. Majd ki kell választani, hogy feltöltés közben jelezze-e a hibát, ha van. Ez után pár percig eltart míg feltölti adatokkal a táblát. A 12.ábrán

egy sikeresen hozzáadott hdi Container látható, benne az Ingatlanok táblával, amiben az adatok vannak töltve.

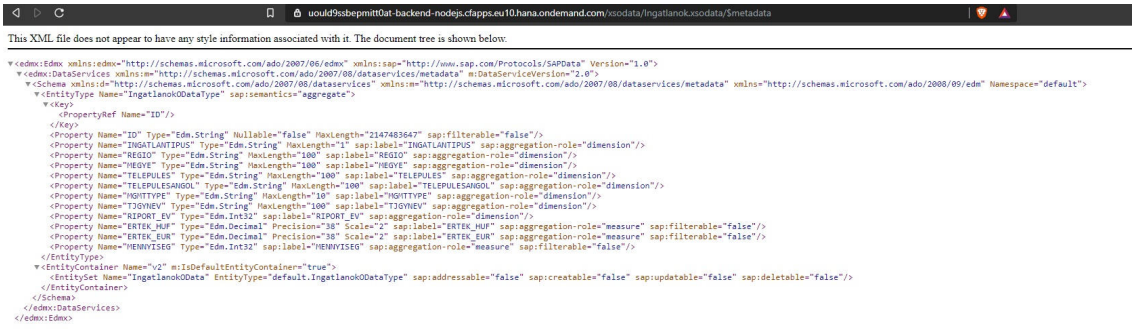


12. ábra. A hdiDB táblái.

Vissza kell térni a bal oldali menüsor Development részébe, majd a backend projekt NodeJS mappájára kattintani jobb egérgombbal és a Run - Run as Node.js Application parancsot választani. Ez a folyamat kicsit több időt vesz igénybe(3-5 perc), amikor elindul a konzol ablakban megjelenik egy link. Másolni kell a linket, ami a 13.képen látható.

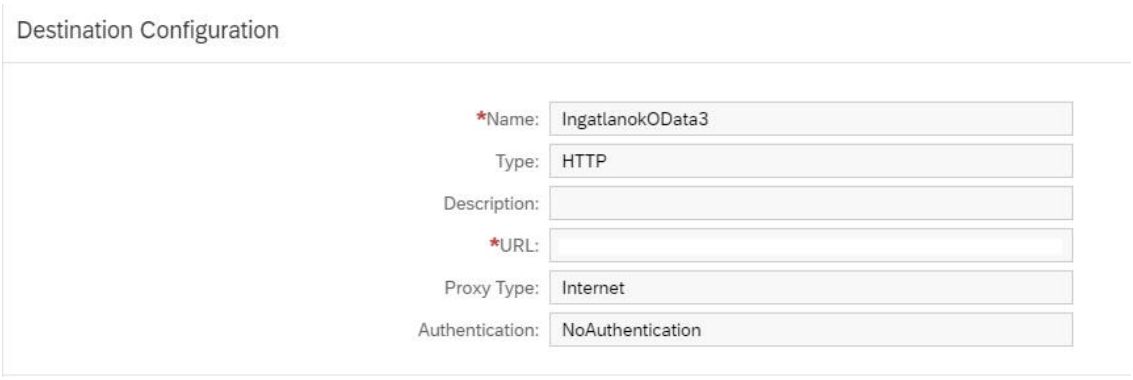
13. ábra. Node.js modul által generált link.

A linket megnyitva lelehet ellenőrizni a Node.js modul működését, kiegészítve még a megnyitott linket a következő képpen:
Az xjsj helyére a következő szöveget kell beírni:
/xsodata/Ingatlanok.xsodata/$metadata
Ezzel a Node.js modul által publikált adatok metaadatait jelenítette meg xml formátumban(14.ábra).



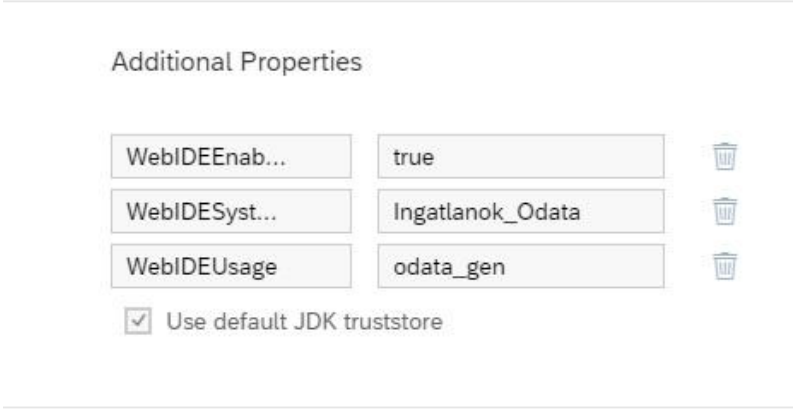
14. ábra. Az ingatlanok metaadatatit ábrázoló xml.

Kilépve a webes fejlesztői környezetből, az SAP Cloud Platform Cockpit főoldalon az "Enviroments" címsornál kell kattintani az "Access Neo Trial" gombra. Az új oldal baloldali menüjében a "Connectivity", azon belül a "Destinations" lehetőséget kell kiválasztani. Majd létre kell hozni egy új célállomást("New Destination"), ami az adatok publikálásáért felel, a következő konfigurációval(15.ábra):



15. ábra. A célállomás konfigurációja.

Az URL szövegmezőbe pedig a webes fejleszetői környezetből kimásolt linket kell beilleszteni. Az Additional Porperties részhez adjunk hozzá 3 új tulajdonságot(New property), amiket a következő képpen állítunk be(16.ábra a következő oldalon):



16. ábra. A 3 új tulajdonság beállításai.

Mindezek után visszalépve a fejlesztői környezetbe a frontend projekt elindítható jobb egérgombbal rákattintva és a Run - Run Index.html parancsot kiválasztva.

2.2.2. Rendszerkövetelmények

A rendszer mivel teljes egészében az SAP Cloud Platform webes felületén lett létrehozva és használatához is elengedhetetlen, ezért szinte platform független, amit a következő táblázat[1] jól ábrázol a 17.ábrán:

Platform	Device Category	Platform Version	Safari	Web View	Internet Explorer	Microsoft Edge	Google Chrome	Mozilla Firefox	SAP Fiori Client
Windows	Desktop	Windows 7, 8.1	-	-	Version 11[2]	-	Latest version	Latest version and Extended Support Release (ESR)[1, 5]	-
		Windows 10	-	Latest version		Latest 2 versions			-
	Touch[3, 4]	Windows 10	-	Latest version	Version 11[2]	Latest 2 versions[5]	Latest version[5]	Latest version and Extended Support Release (ESR)[5]	Latest version
macOS	Desktop	Latest 2 versions	Latest 2 versions	-	-	-	Latest version [3]	-	-
iOS	Phone and Tablet[3, 4]	Latest 2 versions	Latest 2 versions	Latest version[5]	-	-	-	-	Latest version[5]
Android	Phone and Tablet[3, 4, 5]	As of version 5	-	-	-	-	Latest version	-	Latest version

17. ábra. Böngészők és platformok, amiket a rendszer támogat.

A rendszer működtetéséhez szükséges internet kapcsolat, illetve SAP Cloud Platform felhasználói fiók.

3. Fejlesztői dokumentáció

3.1. Követelmények

Magyarország ingatlanjainak nagy mennyiségű adatainak összetett hierarchiák mentén történő aggregálása és az aggregált adatok publikálása az adatbázisból OData szabványnak megfelelő RestAPI interfészen Node.js alapú technológiával(szabványos .json és .xml formátumban) backend oldalon. Frontend oldalon ezen publikált aggregált adatok megjelenítése különböző vizualizációs elemekkel, mint pl. térkép, táblázat, diagram. A frontend oldali elemek AngularJS architektúrában kerülnek implementálásra. A munkafolyamat során az SAP Cloud Platform környezetét használom.

3.2. Megvalósítás folyamata

Kezdetben egy .csv(Comma-separated values) formátumú fájlból indultam ki, ami Magyarország településenkénti ingatlan adatait tartalmazta. A .csv fájl egy sora a következő adatokat tartalmazta:

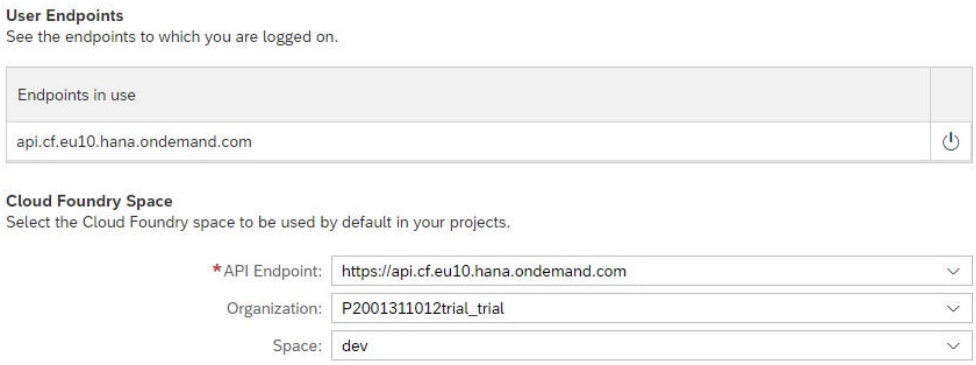
- Érték forintban
- Érték euróban
- Ingatlan típusa
- Ingatlanok mennyisége
- Régió
- Megye
- Település
- MGMTTYPE
- Tárgy jegyző neve
- Riport év

A munkát az SAP Cloud Platform környezetében csináltam. Létrehoztam egy SAP Cloud Platform felhasználói fiókot, majd bejelentkezés után A Cloud Foundry Trial részben létrehoztam a saját Cloud Foundry területemet az Európai szerverükön, amit használtam majd a fejlesztői környezeten belül. Ezután a munkát az SAP Cloud Web IDE környezetében folytattam.

A Web IDE környezetbe való belépés után beállítottam a következő kiegészítőket(Extensions) a "Preferences/Extensions" oldalsó fülön belül, amiket majd később a munkafolyamat során felhasználtam az egyes modulok létrehozásakor:

- OData Model Editor
- SAP HANA Database Development Tools
- Tools for Node.js Development
- SAP HANA Database Explorer
- Annotation Modeler
- Extension Development for SAP Web IDE
- SAP Cloud Platform Services
- Tools for Java Development

A kiegészítők beállítása után a Cloud Foundry beállítása következett, mivel ez felel a webes fejlesztői környezetben a fájlok fordításáért és futtatásáért. Itt beállítottam a nemrég létrehozott Cloud Foundry területem(18.ábra), majd mentettem a változtatásokat.



18. ábra. A Cloud Foundry beállítása.

3.2.1. Backend

Létrehoztam a saját munkaterületemben(workspace) egy új MTA(Multi Target Application) projektet. Ez az MTA projekt tartalmazza majd a backend rész megvalósítását. Következett az adatbázis létrehozása. Az MTA projekten belül csináltam egy SAP HANA Database Modul-t(adatbázis modul), ami a 2.0 SPS 03 adatbázis verziót tartalmazta. Ahhoz, hogy az adatbázis modul fordítható legyen, az src mappán belül található .hdiconfig fájl első sorában a plugin verziót(plugin_version)

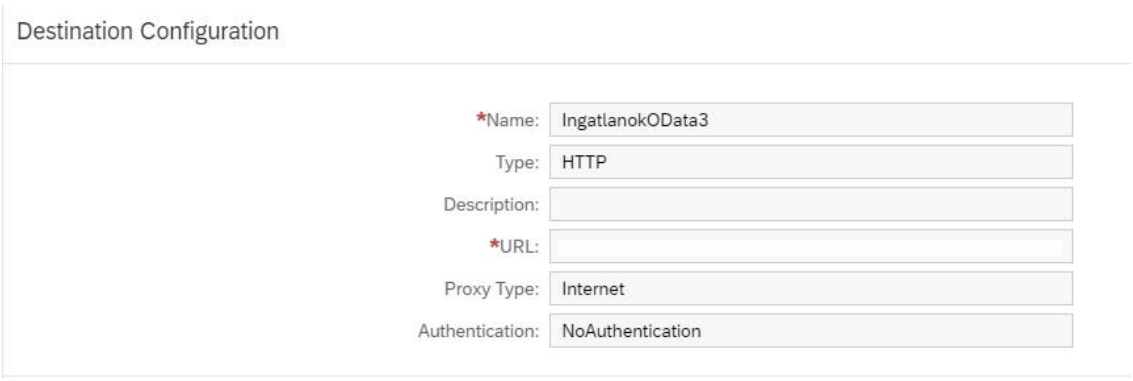
2.0.2.0-ra kellett módosítani. Az src mappában létrehoztam egy HDB CDS Artifactot. Ebben a .hdbcds kiterjesztésű fájlban definiáltam egy új entitást, amely a következő attribútumokat tartalmazta:

- ERTEK_HUF
- ERTEK_EUR
- INGATLANTIPUS
- MENNYISEG
- REGIO
- MEGYE
- TELEPULES
- TELEPULESANGOL
- MGMTTYPE
- TJGYNEV
- RIPORT_EV

Az entitás definiálásakor ügyelni kell rá, hogy az entitás neve és a .hdbcds kiterjesztésű fájl neve megegyezzenek, különben a fordító hibát jelez. Mindezek után lefordíttattam az egész Database modult. Átléptem a Web IDE Database Explorer részébe, ahol hozzáadtam az előbb létrehozott és lefordított adatbázist a Database Explorer-hez. Miután itt is létrejött az adatbázis, a táblái(tables) között már szerepelt az, amit a .hdbcds fájlban entitásként definiáltam. Ebbe a táblába beimportáltam a .csv fájl tartalmát, a különböző megfeleltetéseket, hogy a tábla melyik oszlopába melyik érték kerül, azt a környzet automatikusan kitalálta az oszlopnevek alapján.
A Web IDE Development részén a Database modul src mappában csináltam egy új Calculation View-ot. Ebben a fájlban az adatbázis adatok aggregálásának tulajdonságai kerültek definiálásra. A bemenő adatok a nemrég feltöltött Ingatlanok tábla adatai, a kimenet pedig egy olyan tábla, ahol a követekező oszlopok összegzés alapján aggregálva vannak:

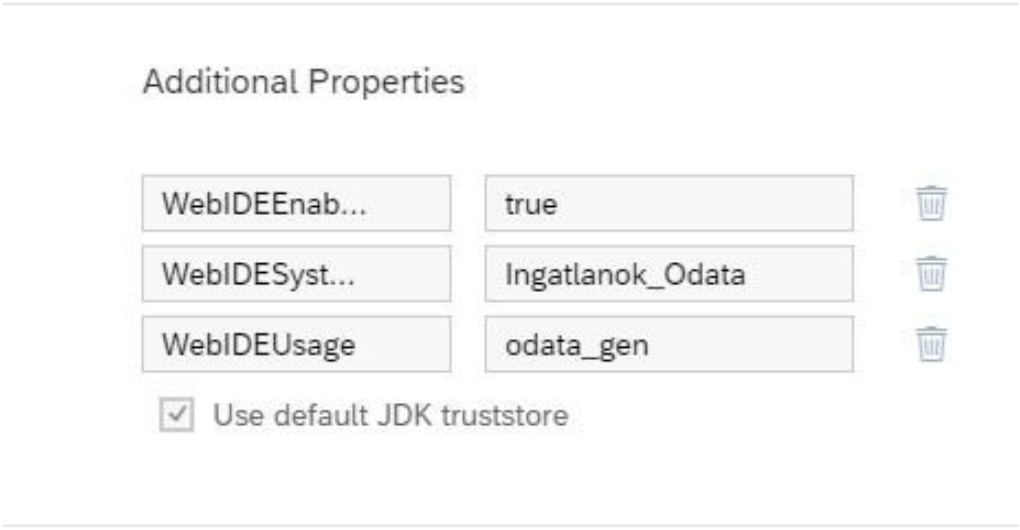
- ERTEK_HUF
- ERTEK_EUR
- MENNYISEG

Következőleg szintén az MTA projekten belül létrehoztam egy Node.js modult. A modul lib mappájába került egy új xsodata mappa, amin belül létrehoztam egy .xsodata kiterjesztésű fájlt. A fájlban definiáltam egy service-t ami az adatbázis aggregált adataiból OData entitást csinál. Futtattam a Node.js modult, ami generált egy linket, amin kerszertül már elérhetőek voltak az adatbázis aggregált adatai.
Létre kellett hozni egy új Destination-t az SAP Cloud Platform főoldaláról elérhető Neo Trial részén. Ez felelt az OData entitás publikálásáért. Az új destination-t a következő képpen konfiguráltam(19.ábra):



19. ábra. A célállomás konfigurációja.

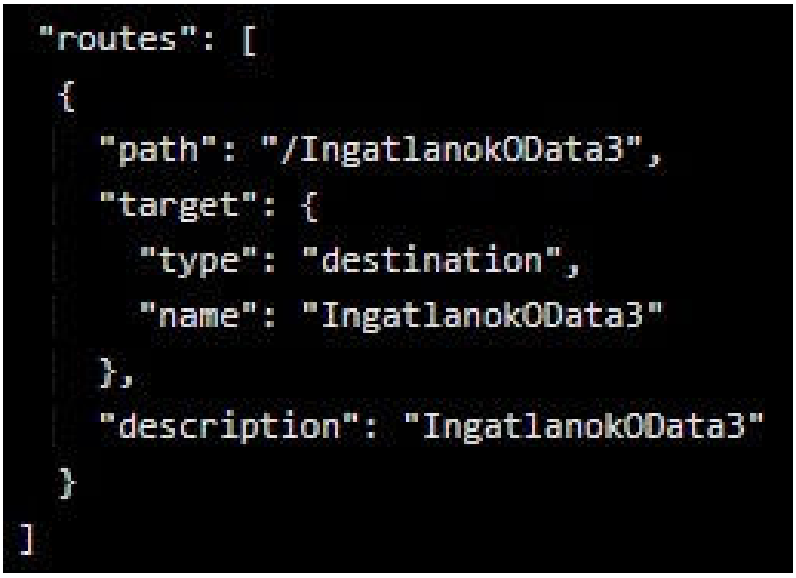
Az Url szövegmezőbe a Node.js modul által generált linket másoltam be. Hozzáadtam 3 új tulajdonságot(property) a következőképp(20.ábra):



20. ábra. Az új tulajdonságok konfigurációja.

3.2.2. Frontend

A saját munkaterületemen létrehoztam egy mapppát, ez tartalmazza a frontend rész fájljait. Megírtam egy neo-app.json fájlt, amiben az adatokat publikáló destination elérését definiáltam(21.ábra).



21. ábra. A neo-app.json konfigurációja.

Következőleg létrehoztam az index.html, main.js és styles.css fájlokat. Először a publikált adatok elérése volt a célom, tehát definiáltam egy odataUrl változót, aminek a publikált adatok elérési utvonalát adtám át értékként, majd egy GET kérésnek átadtam ezt a változót paraméterül, ahol válaszként megkaptam a publikált OData adatokat.
Amikor sikerült elérnem az adatokat, akkor a táblázati és diagramos megjelenítés következett. Egy külső könyvtárat használtam fel ezekhez a vizualizációs elemek megvalósítására. A Devexpress külső könyvtár PivotGrid eleme megfelelőnek tűnt az adataim megjelenítéséhez, mivel könnyen össszeköthetőnek tűnt az ugyanebből a könyvtárból használt diagrammal. A $scope-ban példányosítottam egy DevExpress.data.PivotGridDataSource objektumot dataSource néven úgy, hogy paraméterként megadtam az adatmezők struktúráját, illetve a változót amiben már az adatok voltak. Ezután definiáltam a diagram(chartOptions) és táblázat(pivotGridOptions) tulajdonságait(property). A táblázat onInitialized property-ben kötöttem össze a diagrammal. Majd a táblázat dataSource propertyjében átadtam a nemrég definiált $scope.dataSource-t. Ezzel már a táblázat és a diagram is felvolt töltve adattal. A diagram és a táblázat közötti párhuzamos működés már egyirányban működött(a táblázatban a kattintás hatással van a diagramra, fordítva még nem).

Abban az esetben, hogy fordítva is működjön, azaz kattinthatóak legyenek a diagram elemei, ezért definiáltam a chartOptions onPointClick property-jét(22.ábra). Egy seriesWords tömb változóban tároltam a kattintott elem nevét szóközönként szétdarabolva a split() függvény használatával. Majd megnéztem, hogy milyen hosszú ez a seriesWords tömb, ami alapján eldönthető volt, hogy a táblázat melyik celláját kell lenyitni. A lenyitást a pivotGridOptions.dataSource.expandHeaderItem() függvényével oldottam meg átadva paraméterül, hogy milyen típusú mező lesz lenyitva(row) és magát a mező nevét. Tömbben kellett átadni a mező nevét, mivel ha nem az első, hanem esetleg a második szintet, azaz megye szintet szeretnénk lenyitni, akkor a régió és megye mező nevét is át kell adni.

```
onPointClick: function (e) {
    var seriesWords = e.target.data.series.split(' ');
    var arr = [];
    if(seriesWords.length < 9) {
        if (seriesWords.length <= 4) {
            arr.push(seriesWords[0] + " " + seriesWords[1]);
            displayRegionOnMap(seriesWords[0] + " " + seriesWords[1]);
        } else {
            arr.push(seriesWords[0] + " " + seriesWords[1]);
            arr.push(seriesWords[3] + " " + seriesWords[4]);
            selectCounty(seriesWords[3] + " " + seriesWords[4]);
        }
        $scope.pivotGridOptions.dataSource.expandHeaderItem("row", arr);
```

22. ábra. A diagram onPointClick property-je.

Definiáltam egy dataLoaded változót a $scope-ban, ami kezdetben hamis értéket kapott, majd amikor betöltődtek az adatok, akkor igazra lett állítva. Ez a változó azért volt szükséges, mert ha a táblázat és a diagram hamarabb jelentek meg, mint az adatok betöltődtek volna, akkor üresen, adat nélkül jöttek létre. Ezért a html fájlban a betöltés előtt egy ng-if elágazással szabályoztam, hogy csak az adatok betöltődése után jelenjenek meg a táblázat és a diagram(23.ábra). Ekkor már működött párhuzamosan a táblázat és a diagram.

```
<div id="pivotgrid-demo" ng-if="dataLoaded">
    <div id="chart" dx-chart="chartOptions"></div>
    <div id="pivotgrid" dx-pivot-grid="pivotGridOptions"></div>
    <div dx-popup="popupOptions">
        <div data-options="dxTemplate: { name: 'content' }">
            <div dx-data-grid="dataGridOptions"></div>
        </div>
    </div>
</div>
```

23. ábra. A html kódban az ng-if elágazás.

A térképes nézet megvalósításához a Google Maps Javascript API-ját és a Geocoding API-ját használtam fel. A Maps Javascript API maga a térképet biztosítja. Ahhoz, hogy használni tudjam a Google API-jait, ehhez kérnem kellett egy API kulcsot, amivel működtetni tudtam a térképet az oldalamon. A térképen, hogy kitudjam jelölni Magyarországot megyénként vett felosztásban, megkerestem Magyarország megyéinek Geojson adatait. Minden megye Geojson adatát kiegészítettem egy id tulajdonsággal, aminek a megye hivatalos rövidítését adtam át. Példányosítottam egy google.maps.Data() objektumot stateLayer néven, aminek a loadGeoJson() metódushívással átadtam az általam átszerkesztett Geojson adatot paraméterként megadva a Geojson elérési útvonalát. A térképen a megyék megjelenését a stateLayer setStyle metódusával állítottam be. Ahhoz, hogy a térképen megjelölt megyékre kattintani is lehessen, hozzá kellett adni a stateLayer-hez egy Listener-t, hogy kattintást észlelve a kijelölt megyén beszínezze azt. A festéshez a stateLayer revertStyle() és overrideStyle() metódusait használtam. A stateLayer-t össze kellett kötni egy térképpel, mivel ez magában csak egy réteg a térképen. Tehát példányosítottam egy google.maps.Map objektumot az általam megírt beállításokkal(mapOptions), majd stateLayer setMap metódushívásával összekötöttem az előbb létrehozott térképpel. A térképre való kattintáskor a kattintott elem feature adattagjának getProperty() metódusával lekérdeztem a régió és a megye nevét, amit két külön változóba el is mentettem. Ezt két változót beleraktam egy tömbbe, majd meghívtam a pivotGridOptions.dataSource.expandHeaderItem() metódusát szintén a cellatípus(row) és a tömb paraméterekkel. Már a térképen kiválasztva egyik megyét hatással van a táblázatra és a diagramra is. Ahhoz, hogy a táblázaton vagy a diagramon kattintva hatással legyen a térképre, definiáltam két függvényt:

- displayRegionOnMap(region)
- displayCountyOnMap(county)

Ezek a függvények kijelölik a térképen a paraméterül kapott régió megyéit vagy a megyét. A táblázaton vagy a diagramon már település szinten kiválasztva egy települést, a térképen ránagyít a településre és megjelöli annak helyét egy jelölővel(marker). Ezt a Google Geocoding API felhasználásával sikerült elérnem, példányosítottam egy google.maps.Geocoder objektumot, majd a táblázat és a diagram onCellClick és onPointClick property-jeiben levő függvényben, ha településre lett kattintva, akkor a település nevét átadtam a Geocoder address-ének, kikerestem a térképen azt, ráközelítettem és elhelyeztem a helyen egy jelölőt(marker).
Az oldalon lévő Alap nézet feliratú gombhoz hozzáadtam egy eseménykezelőt, ami a kattintás hatására visszaállítja a térképet az egész Magyarország nézetére, illetve meghív egy DeleteMarkers() eljárást, ami letörli a jelölőket(marker) a térképről.

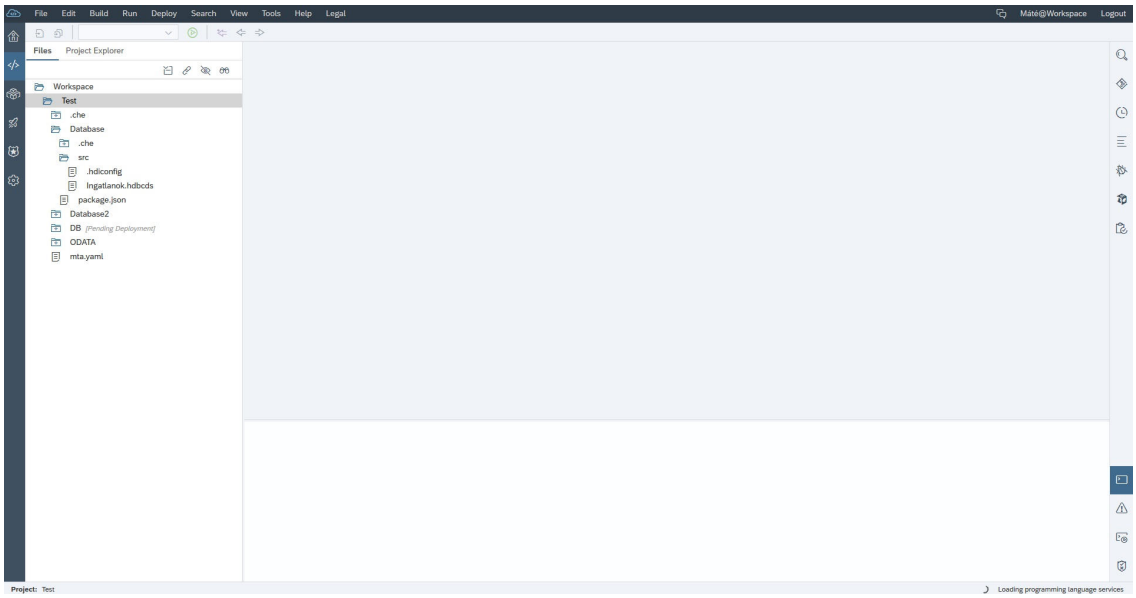
3.3. Használt szoftverek, fejlesztőeszközök és könyvtárak

3.3.1. Dokumentáció

A szakdolgozat dokumentálása során felhasználtam egy előre elkészített Latex környzetben íródott mintát[2], ami tartalmazta a címoldalnak előírt követelmények megvalósítását, illetve az egész dokumentációra vonatkozó alap kitételeknek eleget tevő beállításokat tartalmazott.

3.3.2. Fejlesztői környezet

A megvalósításhoz az SAP Cloud Platform webes fejlesztői környezetét használtam(24.ábra), ami egyaránt lehetőséget kínál adatbáziskezelésre, webes applikáció készítésére, az adatok publikálására RestAPI interfészen keresztül OData szabványnak megfelelően, illetve rengeteg más dologra. Az SAP Cloud Platform két részből áll, a Neo Trial és a Cloud Foundry. A WebIDE a Neo Trial része, de a fordító, amit használ, az a Cloud Foundry része. Nagyon sok előre definiált template projektet és modult is tartalmaz, amik megkönnyítik a fejlesztői folyamatot. Bár az applikáció frontend részéhez nem tartalmazott AngularJS architektúrájú template projektet, tehát manuálisan kellett az egész projektet létrehoznom, de a backend oldalon felhasználtam a környezet nyújtotta MTA(Multi Target Application) projekt-et, Database modult és Node.js modult is.



24. ábra. Az SAP Cloud Platform webes fejlesztői környezete.

3.3.3. AngularJS

Az applikáció frontend részének az implementálására AngularJS-t használtam, ami egy JavaScript alapú nyílt forráskódú fornt-end oldali webes keretrendszer, amit a Google fejleszett. Az AngularJS az Angular keretrendszer első verzióját jelöli. Sok megoldási lehetőséget kínál főleg az egy-oldali alkalmazások(single-page applications) fejlesztésében. Kliens oldali megjelenítésre az MVC(model-view-controller) és az MVVM(model-view-viewmodel) architektúrákat támogatja. A nézetek a HTML fájlok, a kontrollerek és a modellek JavaScript nyelven íródnak. Maga a keretrendszer egy scope objektumot használ, ami az MVC architektúrában a modelt formálja meg. A scope-ban definiált változók elérhetők a kontrollerek és nézetek számára egyaránt. Amikor egy kontrollerből új példány jön létre, akkor egy új gyerek scope is keletkezik.

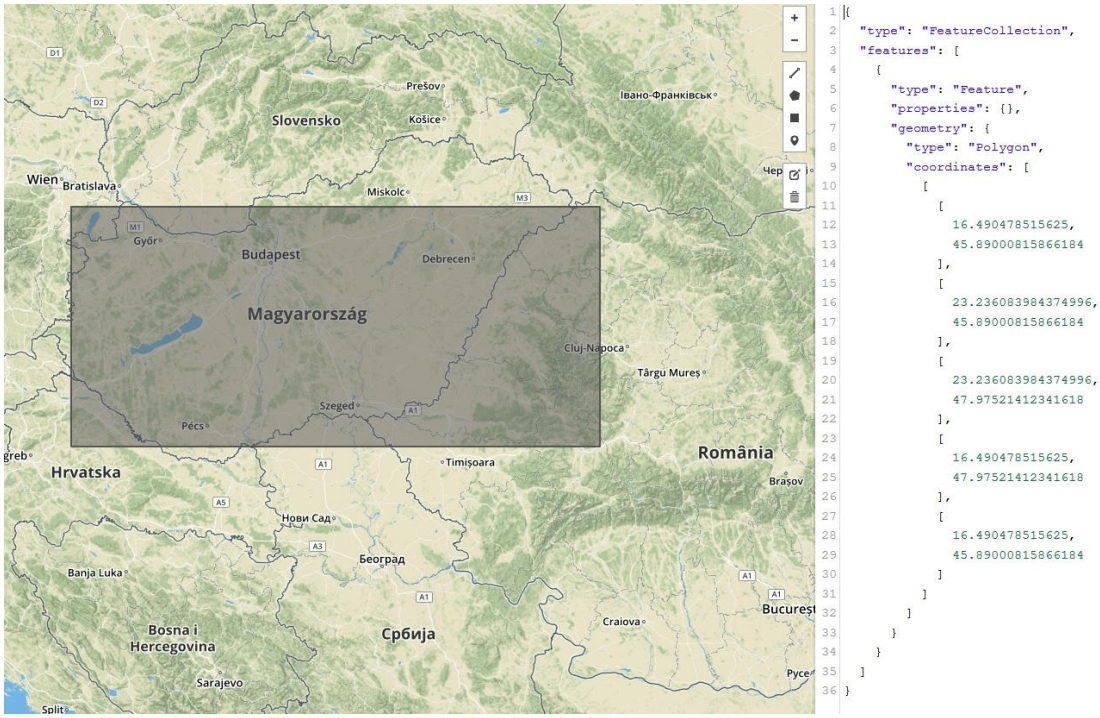
3.3.4. Google Maps Javascript API

A térkép vizualizációhoz a Google API-jai közül a Maps JavaScript API-t és a Geocoding API-t használtam fel.
A Maps JavaScript API javascript környezetben implementálható térképet biztosít, illetve különböző látványelemek is létrehozhatók a térképen a segítségével, ilyen például a különböző rétegek, vagy a jelölők(markers) stb. Az API google.maps.Data osztályát használtam arra, hogy betöltve a saját GeoJSON adataim megjelenítsem azt a térképen. Ennek az osztálynak több metódusát is felhasználtam, ahhoz hogy szerekeszteni tudjam a térképen kijelölt részeket. Ilyen metódusok pl. setStyle(), overrideStyle(), revertStyle().
A Google Geocoding API-ja egy olyan szolgáltatás, amivel geokódolni(geocoding) és fordított geokódolni tudunk. A geokódolás egy olyan folyamat, amikor egy címet(pl. házszám) geográfiai adatokra konvertáljuk, azaz hosszúsági és szélességi körökre, amit már így megtudunk jelölni a térképen. A fordított geokódolás annyit jelent, hogy geográfiai adatokat(hosszúsági és szélességi körök) alakít valós címekre. Én a geokódolás részét használtam fel, amikor a felhasználó egy településre kattint a táblázatban vagy a diagramon, akkor a település geokódolva van, majd a geográfiai koordináták alapján megjelöltem a térképen.

3.3.5. GeoJSON

A térképes nézethez felhasználtam Magyarország megyéinek GeoJSON adatait is[3]. A GeoJSON egy olyan formátum, amellyel geológiai területeket reprezentálnak JSON alakban, amely többek között a terület azon koordinátáit tartalmazza, amellyel a földrajzi terület határolva van(25.ábra következő oldalon). Ez a GeoJSON formátum a JSON szabványon alapszik, amely egyszerű adatstruktúrák

és asszociatív tömbök reprezentálására szolgál. Ezek a GeoJSON adatok különböző geometriai alakzatokat ábrázolhatnak, mint pl. pont, szakaszok, poligon. Az általam használt GeoJSON poligonok pontjainak koordinátáit tartalmazza, így leírva Magyarország minden megyéjének kerületét. Tartalmazza azt is, hogy melyik megye, melyik régióba tartozik. Hozzá kellett adnom egy id tulajdonságot is, amiben az ISO 3166-2:HU szabványban feltüntetett földrajzi kódjait tároltam a megyéknek[4].

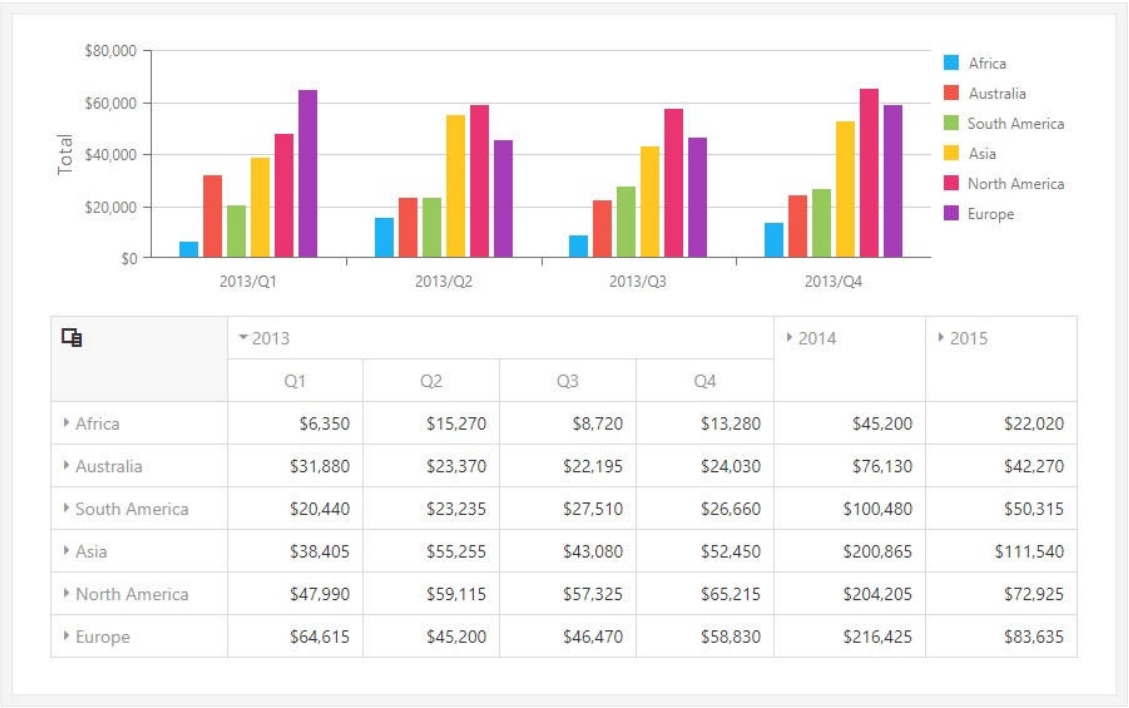


25. ábra. Példa a GeoJSON-re.

3.3.6. Táblázat és diagram

A táblázatos és diagramos megjelenítéshez egy külső DevExtreme könyvtárat használtam fel, amely nagyon sok fajta adatvizualizációs eszközt tartalmaz[5]. Ezek közül én a PivotGrid és egy oszlopdiagramos megjelenítési formát használtam, mivel ezek az elemek támogatják az adatok közti lefúrást(drilldown), könnyen összeilleszthetők, mindezek mellett pedig nagyon látványosak és formálhatóak. A PivotGrid egyfajta táblázat(26.ábra), amely tartalmaz lenyitható sorokat a lefúrás mentén, így sikerült egy olyan táblázatot létrehoznom, ahol lenyitva valamelyik régiót megjellenek a régióba tartozó megyék, megyét lenyitva pedig a bele tartozó települések. A táblzat ezen mezőit sor típusú celláknak jelölik, a mellettük összegzett adat pedig az adat típusú mezők. Ahhoz, hogy a táblázat és a diagram párhuzamosan működjön, a pivotgrid inicializáslásakor csak összekellett kötni(bind) a diagramal. Az adatokat is

egy DevExpress.data.PivotGridDataSource példányosított objektumban tároltam, ahol példányosításkor megadtam az adatok szerkezetét is.

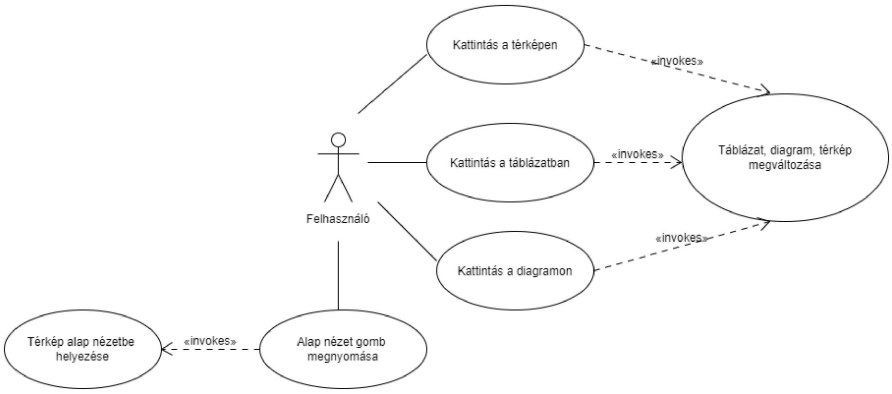


	2013				2014	2015
	Q1	Q2	Q3	Q4		
Africa	$6,350	$15,270	$8,720	$13,280	$45,200	$22,020
Australia	$31,880	$23,370	$22,195	$24,030	$76,130	$42,270
South America	$20,440	$23,235	$27,510	$26,660	$100,480	$50,315
Asia	$38,405	$55,255	$43,080	$52,450	$200,865	$111,540
North America	$47,990	$59,115	$57,325	$65,215	$204,205	$72,925
Europe	$64,615	$45,200	$46,470	$58,830	$216,425	$83,635

26. ábra. Egy minta PivotGrid elemre.

3.4. Felhasználói esetek

A felhasználó ha a táblázat egy lenyitható fülére kattint(Régió vagy megye), akkor az hatással van a térképre és a diagramra is. Ha a diagram egyik elemére kattint, akkor úgyszintén hatással van a táblázatra és a térképre is. A térképen egy megye kiválasztásakor, megváltoznak a táblázatban és a diagramon megjelenítet adatok. A táblázatban vagy a diagramon egy települést kiválasztva a térképes nézet megjelöli a települést a térképen és rá is közelít. Az alapnézet gomb megnyomásakor a térkép az alap nézetbe kerül és letörlődnek róla az elhelyezett jelölők.
Ezeket a felhasználói eseteket a következő oldalon a 27.ábrán látható felhasználói eset diagram ábrázolja.



27. ábra. Felhasználói eset diagram.

3.5. Logikai adatmodell

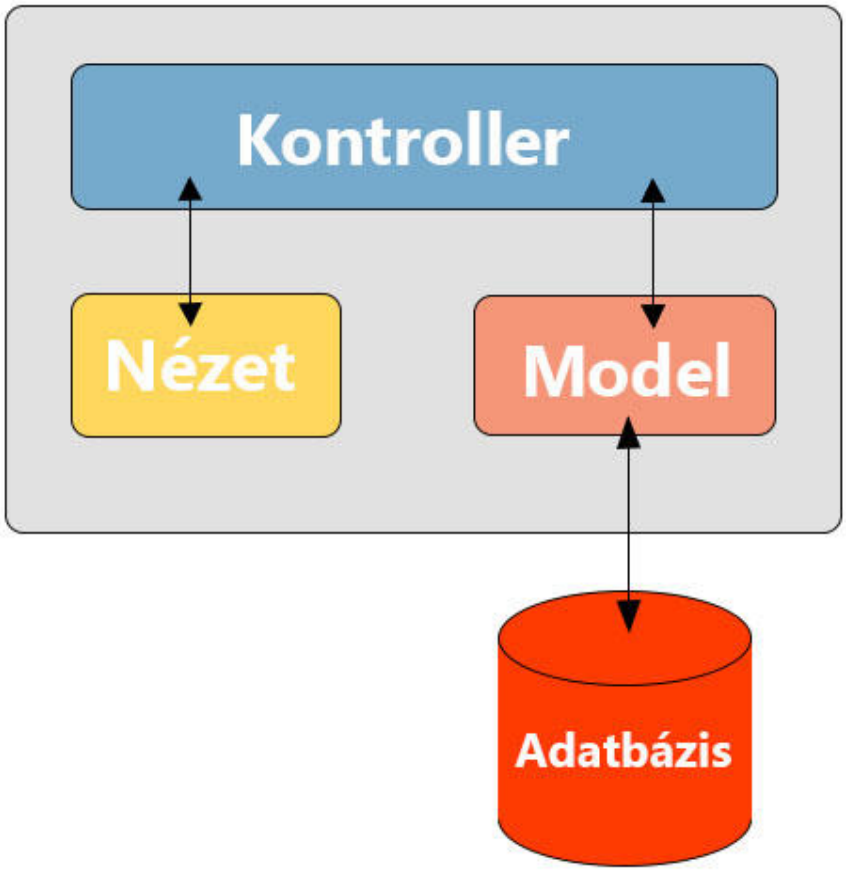
Az Adatbázis az adatokat egy Ingatlanok nevű táblában tárolja, oszlopalapú adattárolási módszert használva. Az oszlopalapú adattárolás annyiban tér el a soralapú adattárolási módszertől, hogy míg a soralapú a tábla sorait tárolja, addig az oszlopalapú a tábla oszlopait tárolja. A következő táblázat egy oszlopalapú adattárolásra mutat példát az ingatlanok adatait felhasználva.

Dél-Alföldi régió	Dél-Alföldi régió	Észak-Magyarország régió	Dél-Dunántúl régió
Csongrád megye	Csongrád megye	Nógrád megye	Somogy megye
Zákányszék	Zákányszék	Márkháza	Bedegkér
2010	2011	2011	2012

Ebben az esetben azért előnyös az oszlopalapú adattárolás, mivel nagy mennyiségű adat lekérdezésénél, illetve aggregálási művelet végrehajtásánál sokkal gyorsabbak. A lekérdezéseknél az előnye abban rejlik, hogy akár párhuzamosan több lekérdezést is végre tud hajtani különböző oszlopokon, mivel az oszlopok vannak tárolva. Az aggregálásnál abból származik az előny, hogy az egyes oszlopokon van az aggregálási művelet végrehajtva, tehát elég azt az egy oszlopot beolvasnia, ellentétben a soralapúval, ahol ehhez az egész táblát be kell olvasni. Az adatok módosításakor a soralapú az előnyösebb, de mivel az adatbázisban a beimportált ingatlanok adatot nem változtatom, ezért előnyösebb sokkal az oszlopalapú tárolás. Az adatbázis adatai még megosztás előtt aggregálva vannak, szummázva az érték_HUF, érték_EUR és MENNYISEG oszlopok alapján. A frontend oldali felhasználáskor kerülnek csak az adatok hierarchikus formába. A weboldalon a táblázat és a diagram már ezt a hierarchia szerint szervezett adatokat jelenítik meg.

3.6. A program architektúrája

A program két fő részből áll össze, a backend, ami az adatbázis részét jelenti, illetve a frontend, ami egy webes felületet biztosít. A frontend rész megvalósítása a model-nézet-vezérlő(MNV, angolul model-view-controller) architektúrát követi, ezzel több rétegre bontva az alkalmazást.
A model, amit jelen esetben AngularJS környezetben a $scope formál meg, az adatok kezeléséért felel, ebben tárolódik az adatbázisból elért adat.
A nézet felel az adatok megjelenítéséért, ami egy HTML oldal fromájában lett megvalósítva.
A vezérlő, más néven kontroller kezeli a nézettől kapott kéréseket, illetve az adatokkal való műveleteket. A 28.ábra az MNV architektúra működését mutatja be.



28. ábra. MNV architektúra.

3.7. Függvények leírása

3.7.1. displayCountyOnMap

A függvény feladata, hogy a térképen megjelenítse a kiválasztott megyét, illetve ha már megvolt nyitva a megye, akkor letörli a térképen a megye kiszínezését és azt a régiót jelöli ki a térképen, amelybe a becsukott megye tartozik.
Paraméterei:

- county - a megye ID-ja
- countyNum - a megye sorszáma a selectedCounty tömbben
- regionNum - a régió sorszáma a selectedRegion tömbben, amelybe a megye tartozik

A függvény törzsében egy elágazás található, amely eldönti a countyNum paraméter felhasználásával, hogy a megye a táblázatban lenyitva vagy bezárva lett. Ha lenyitva, akkor kiszínezi a térképen a megyét, ellenkező esetben letörli a megye kiszínezését, meghatározza a regionNum paraméter alapján, hogy mely megyék tartoznak az adott régióba, majd kiszínezi ezeket a megyéket.

3.7.2. displayRegionOnMap

A kiválasztott régión belüli megyék megjelenítéséért felel, ha először lett kattintva a régióra, amennyiben előtte már lenyitva volt, akkor letörölnie kell a kijelölést a térképen.
Paraméterei:

- region - a kiválasztott régió neve

A függvény először a region paraméter alapján egy többágú elágazásban beállítja a regionNum változó értékét. Ezután egy elágazásban eldönti, hogy a régió lenyitva lett, vagy visszacsukva. Ha lenyitva lett a régió, akkor a regionNum értékét felhasználva kiszínezi a térképen a régióba tartozó megyéket. Ha visszacsukva lett, abban az esetben letörli a térképről a kijelölt régióba tartozó megyék színezését.

3.7.3. selectCounty

A függvény feladata, hogy a paraméterül kapott megye név alapján meghívja a displayCountyOnMap függvényt átadva neki a megfelelő paramétereket.
Paraméterei:

- county - a kiválasztott megye neve

Egy több ágú elágazásban a county paraméter alapján felparaméterezi a displayCountyOnMap függvényt, átadva neki a paraméterül kapott megye ID-ját, sorszámát a selectedCounty tömbben és a régió, amelybe tartozik a megye annak sorszámát a selectedRegion tömbben.

3.8. Tesztelés

A program tesztelését a használati esetek alapján végeztem. Összehasonlítottam a felhasználói behatás után elvárt eredményt a bekövetkezett eredménnyel.

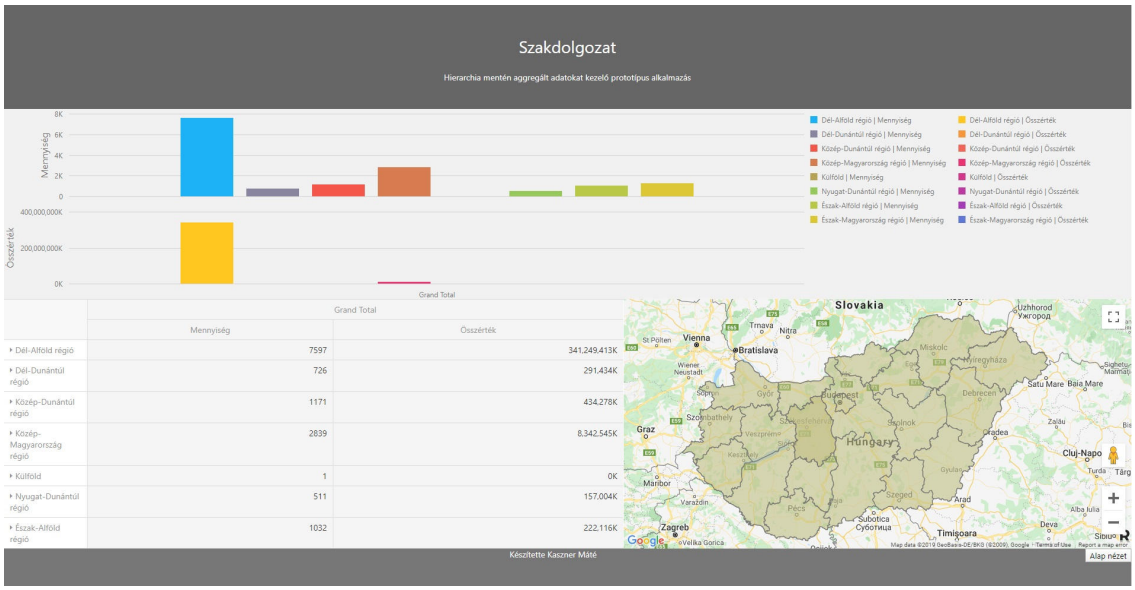
Felhasználói eset	**Várt eredmény**	**Eredmény**
Kattintás a táblázatban lenyitható elemre	Lenyílik a táblázat sora, a térkép és a diagram is megjeleníti a lenyitott elemet	Lenyílt a táblázat sora, a térkép és a diagram is megjelenítette a lenyitott elemet
Kattintás a táblázatban becsukható elemre(Régió)	Becsukódik a táblázat sora, a diagram is a régiókat ábrázolja, a térképen megszűnik a bezárt régió kijelölése	Becsukódott a táblázat sora, a diagram is a régiókat ábrázolja, a térképen megszűnt a bezárt régió kijelölése
Kattintás a táblázatban becsukható elemre(Megye)	Becsukódik a táblázat sora, a diagram a becsukott megyével egy régióba tartozó megyéket ábrázolja, a térképen a régió megyéi rajzolódnak ki, amibe a kiválasztott megye tartozik	Becsukódott a táblázat sora, a diagram a becsukott megyével egy régióba tartozó megyéket ábrázolja, a térképen a régió megyéi rajzolódtak ki, amibe a kiválasztott megye tartozik
Kattintás a diagram egy elemére(Régió, megye)	Az elemen belüli adatok megjelennek a diagramon, lenyílik a táblázatban és kirjazolódik a térképen a kiválasztott elem	Az elemen belüli adatok megjelentek a diagramon, lenyílt a táblázatban és kirjazolódott a térképen a kiválasztott elem
Térképen egy megye kiválasztása	A táblázat és a diagram is ábrázolja a kiválasztott elemet	A táblázat és a diagram is ábrázolta a kiválasztott elemet

Felhasználói eset	**Várt eredmény**	**Eredmény**
Táblázatban kattintás településre	A térkép megjelöli a települést és ráközelít	A térkép megjelölte a települést és ráközelített
Alapnézet gomb megnyomása, ha a térképen van megjelölt település	A térkép visszaáll a kezdeti nézetbe és letörli a jelölőket	A térkép visszaállt a kezdeti nézetbe és letörölte a jelölőket
Alapnézet gomb megnyomása, ha a térképen nincs megjelölt település	Nem történik a térképen semmi	Nem történt a térképen semmi

4. Összefoglalás

Feladatom az SAP Cloud Platform webes környezetében egy hierarchia mentén aggregálható adathalmaz feldolgozása, illetve a feldolgozott adat látványos megjelenítése volt. A megjelenítése webes felületen, különböző vizualizációs eszközök felhasználásával történt. Ezek közé tartozik a lenyitható táblázat, oszlopdiagram és a térkép. A weboldalon elhelyezett megjelenítő eszközök elsődleges célja az volt, hogy a hierarchia mentén történő lefúrást vizualizálni tudják, azaz szintekre bontva(régió, megye, település) ábrázolják Magyarország településenkénti ingatlanokkal kapcsolatos adatait.
A megoldás során a projektet két főbb részre osztottam, a backend és a frontend részekre. A backend az adatbázist, az adatok feldolgozását, aggrelását, illetve az aggregált adatok OData szabvány alapján való publikálását tartalmazta. A frontend részhez a webes alkalmazás tartozott, ami AngularJS környezetben, model-nézet-vezérlő(MNV) architektúrában lett implementálva.
A fejlesztést a backend résszel kezdtem, létrehozva egy MTA(multi target application) projektet, ami későbbiekben az adatbázis és Node.js modulokat tartalmazza. Az adatbázis modulban definiáltam egy ingatlanok entitást, majd ezután a .csv fájlból, ami a kiindulási adatokat tartalmazta, betöltöttem ebbe az ingatlanok entitásba az adatokat. Ezután az adatok aggregálása következett, amit úgynevezett Calculation view segítségével valósítottam meg, az ingatlanok táblából származó adatok érték_huf, érték_eur és mennyiség alapján szummázva. A Node.js modul az adatbázisban lévő, már aggregált adatok megosztását biztosította. A Node.js modult futtatva, generált egy linket, amin keresztül érhetők el az adatok. Egy célállomást(angolul destination) létrehozva publikáltam az adatokat, a konfigurációjában megadva a Node.js modul által generált linket. A célállomás létrehozása azért volt fontos, hogy más projektek is eltudják érni a megoszott adatot.
A frontend részt megvalósító projekthez nem használtam a fejleszető környezetben előre definiált modult, hanem létrehoztam egy mappa szerkezetet, amiben beállítottam a neo-app.json állományban a backend adatait publikáló célállomás elérését. Ezután az MNV architektúrát követve létrehoztam a html és javascript fájlokat. A html fájl a nézetet biztosította, míg a javascript állományban a vezérlő és a model foglaltak helyet. A frontend projekt részeként létrehoztam egy .geojson kiterjesztésű fájlt, amiben eltároltam Magyarország megyéinek GeoJSON adatait. A táblázat és a diagramos megjelenítéshez egy DevExtreme nevű külső AngularJS könyvtárat használtam. A táblázatnak egy PivotGrid elemet példányosítottam, a diagram egy oszlopdiagram formát kapott. A publikált backend adatokat egy Get kérés eredményeként kaptam meg, amit a modelben tároltam. A táblázat és a diagram adat betöltéséhez egy DataSource elemet példányosítottam a $scope-ban, amely már

régió, megye, település szerint hierarchikus módon tartalmazta az adatokat. Ezt az elemet adtam át a táblázat dataSource tulajdonságának, ezzel betöltve az adatokat. A táblázat és a diagram könnyen összeilleszthető volt, a pivotGrid elem onInitialize tulajdonságában kellett csak összekötni a két megjelenítési formát. A térképes nézethez a Google Maps Javascript API és a Google Geocoding API lett felhasználva. A Google Maps Javascript API magát a térképet biztosította, a Geocoding API pedig egy cím geokódolását végezte geográfiai adatokra, azaz hosszúsági és szélességi körök, ami alapján megjelölhető egy pont a térképen. Példányosítottam egy google.maps.Data osztályt, majd beletöltöttem a GeoJSON adatot és ezt ráhelyeztem egy rétegként a térképre. Ahhoz, hogy a táblázattal és a diagrammal párhuzamosan működjön a térkép nézet, saját függvényeket definiáltam, a displayRegionOnMap, a displayCountOnMap és a selectCounty függvényeket, amelyek a paraméter alapján színezik ki a megfelelő megyét/megyéket a térképen.
Sikerült létrehoznom egy olyan alkalmazást, amelynek backend részén az adatok feldolgozásra kerülnek és mindezt frontend oldalon három különböző vizualizációs modul segítségével megtudtam jeleníteni, bemutatva a lefúrásos technikát is. A három megjelenítési rész(táblázat, diagram, térkép) egymástól függően, teljesen párhuzamos működést mutatnak. Az elkészült alkalmazás a 29.ábrán látható.



29. ábra. A kész webes felület.

4.1. Továbbfejlesztési lehetőségek

A legtöbb alkalmazás sohasem éri el a végső állapotát, mert mindig tovább fejleszthető. Ezen az alkalmazáson is vannak további lehetőségek, hogyan lehetne még fejleszteni rajta. A program frontend részén fellehetne használni a többi megosztott

adatok közül valamelyikeket, amiket ugyan csak valamilyen diagram segítségével lehetne ábrázolni, esetleg ezt is összekötve a a már meglévő működő elemekkel. Az adatok beolvasását szintenként lehetne megoldani, azaz míg a táblázat egy eleme sincs lenyitva, addig csak a régiók töltődnének be, majd lenyitva az egyik régiót, csak az adott régióba tartozó megyék töltődnének be és hasonlóan a többi szinten, ezzel csökkentve a betöltés idejét.
Az alkalmazás backend részén amin javítani lehetne, hogy a Node.js modul által több rekord lenne megosztható, mert jelenleg ez korlátozva van 1000 darab rekordra.

5. Irodalomjegyzék

[2]”Latex minta fájl”[Online]. Elérhető
https://github.com/shdnx/ELTE-LaTeX-Thesis-Base
[Hozzáférés dátuma: 2019.12.16.]

[3]”Magyarország megyéinek GeoJSON adata”[Online]. Elérhető:
https://github.com/integralvision/geo-data-hungary/tree/master/GeoJSON/l30-county
[Hozzáférés dátuma: 2019.12.12.]

[4] ”Magyarország megyéinek kódjai, az ISO 3166-2:HU szabvány alapján”[Online]. Elérhető:
https://www.iso.org/obp/ui/#iso:code:3166:HU
[Hozzáférés dátuma: 2019.12.12.]

6. Ábrajegyzék

[1]”Böngésző és platform támogatás táblázat”[Online]. Elérhető:
https://sapui5.hana.ondemand.com/sdk/#/topic/74b59efa0eef48988d3b716bd0ecc933
[Hozzáférés dátuma: 2019.12.12.]

[5]”PivotGrid elem”[Online]. Elérhető:
https://js.devexpress.com/Demos/WidgetsGallery/Demo/PivotGrid/Overview/AngularJS/Light/
[Hozzáférés dátuma: 2019.12.12.]